[HBASE-13408] StoreSegment and StoreSegmentScanner Class Hierarchies

September 24, 2015

Eshcar Hillel (eshcar@yahoo-inc.com)

Anastasia Braginsky (anastas@yahoo-inc.com)

A store is composed of segments. The memstore manages the in-memory segments, while disk segments are represented by HStoreFiles. Default memstore has 2 segments (1) active segment (2) snapshot segment--the previously active segment which became immutable. A compacted memstore has in addition to these two segments a pipeline of segments which are subject to compaction (see MemStore Compaction Design Document). Every type of segment has a scanner that allows to efficiently traverse through the segment's cells.

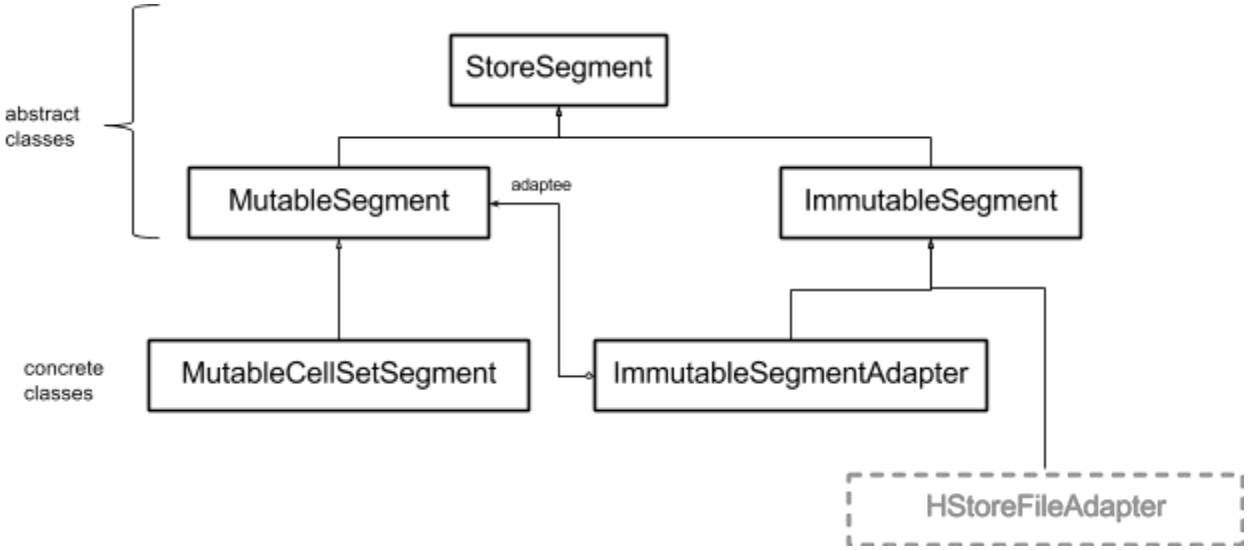
This document focuses on the design and usage of the StoreSegment classes and their respective scanners. We aim to generalize the abstraction captured by a store segment so that a memory segment can be in format different than the skip-list structure. Specifically, we would like to support memory segments which are formatted as HFile. That is, segments which comprise of flat blocks with low memory overhead that can be easily compressed. Each segment has a scanner that is suited to its internal implementation.

StoreSegment class Hierarchy

Figure 1 depicts the StoreSegment class Hierarchy.

Abstract classes

StoreSegment is an abstract segment class. It defines the API which is needed by the memstore to manage the segments.



MutableSegment is an abstract class which extends the API supported by a store segment, and is not needed for immutable segments, e.g., incSize(). Only the active segment is a mutable segment.

ImmutableSegemnt is an abstract class which extends the API supported by a store segment, and is not needed for a mutable segment. Specifically, the method getScannerForMemStoreSnapshot() builds a special scanner for the MemStoreSnapshot object. In addition, this class overrides methods that are not likely to be supported by an immutable segment, e.g., rollback() and getCellSet() which can be very inefficient. Specific sub-classes of immutable segments can re-override these methods if they are able to support them efficiently.

Concrete classes

MutableCellSetSegment is the cell-set based implementation of a mutable memstore segment (refered to as *MemStoreSegment* in the original design document). It encapsulates a mutable skip-list cell set structure and its respective memory allocation buffers (MSLAB).

ImmutableSegmentAdapter wraps and adapts a mutable segment. This is used when a mutable segment is moved to being a snapshot or pushed into a compaction pipeline, that consists only of immutable segments. The compaction pipeline may generate different type of mutable segments.

HStoreFileAdapter TBD. This class wraps an HStoreFile to be used as a memory segment.

Segment Factory

The class **StoreSegmentFactory** is responsible for creating the proper segments. The methods
createImmutableSegment(...)
createMutableSegment(...)
create immutable and mutable segments, respectively. The specific type of the concrete class (as well as other parameters) can be set in the configuration that is passed to the methods.

StoreSegmentScanner class Hierarchy

StoreSegmentScanner is an interface for store segment scanner, can serve both memory segment (memstore segment) and disk segments (files). It extends the KeyValueScanner interface with two additional methods:
setSequenceID() - is needed to determine the order of memory segment scanners when there is more than one (or two);
shouldSeek() - helps to filter out the segments that can be skipped during the scan.
A MemStoreScanner (see the original design document) is executing the scan with a list of store segment scanners, one per segment in the memstore.

MutableCellSetSegmentScanner is currently the only class implementing the store segment scanner interface (referred to as *MemStoreSegmentScanner* in the original design document). It supports an efficient scan of a MutableCellSetSegment.

When HStoreFileAdapter is implemented, the HStoreFileScanner can be refactored to implement StoreSegmentScanner instead of KeyValueScanner, so it can be used as one of the memstore scanners.